VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2013 **Martin Mikula**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Mikula**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c) Zvolený postup řešení zadaných úkolů
d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vedl odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Kot, Ph.D.**

Konzultant bakalářské práce: Kristian Wiglasz

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: 30.4.2013

.....
podpis studenta

Poděkování

Rád bych poděkoval Ing. Martinu Kotovi Ph.D. za odbornou pomoc a konzultaci při vytváření této bakalářské práce, firmě Tieto Czech s.r.o. za příležitost vykonání bakalářské praxe, celému týmu se kterým jsem pracoval a v neposlední řadě své sestře Bc. Veronice Mikulové za korekturu.

Prohlášení zástupce spolupracující právnické nebo fyzické osoby

„Souhlasím se zveřejněním této bakalářské/diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava."

Dne: 30.4. 2013

.....
podpis zástupce

Abstrakt

Tento dokument popisuje průběh mojí praxe ve firmě Tieto Czech s.r.o.

Na začátku práce krátce představuje firmu a mou pracovní pozici. Dále popisuji produkt, na kterém jsem pracoval, jeho architekturu a fungování. V práci je popsáno, jak bylo nad touto architekturou vytvořeno testovací prostředí a unit testy.

Také popisuji generování dokumentace ze zdrojového kódu.

V závěru práce se pokouším ohodnotit, co jsem se během praxe naučil a jaké zkušenosti jsem takto získal.

Klíčová slova

Tieto, C#, .NET, Sandcastle, SHFB, Unit testy, Falešný backend

Abstract

This document describes my activities on practice in Tieto Czech.

First is short presentation of company and my position. Next I am explaining how is working the application I was working on and what is her architecture. I explain how was made testing environment over that architecture and how we made the unit tests. Then I describe how to do generating documentation from source code.

At the end you can find conclusion and evaluation of what I learned and what was that experience.

Key words

Tieto, C#, .NET, Sandcastle, SHFB, Unit tests, Fake backend system

Seznam použitých zkratek

Zkratka	Anglický význam	Český význam
UT	Unit tests	Unit testy
FBS	Fake backend system	Falešný systém na pozadí
TFS	Team Foundation Server	Team Foundation Server
UI	User Interface	Uživatelské rozhraní
SHFB	Sandcastle Help File Builder	Kompilátor nápovědy pomocí nástroje Sandcastle
VS	Visual Studio	Visual Studio
XML	Extensible Markup Language	Rozšiřitelný značkovací jazyk
MVC	Model – View – Controller	Model – Pohled - Řízení

Obsah

1 Úvod

Pro svou bakalářskou práci jsem si vybral odbornou praxi. Rozhodl jsem se tak zejména proto, že je podle mého názoru získání zkušeností, vzhledem k mému zaměření programátora, to nejdůležitější.

Při výběru firmy jsem se zaměřil na ty, které mi nabízely práci s mými oblíbenými technologiemi, a tedy Microsoft .NET. Na základě přijímacího pohovoru jsem byl přijat do společnosti Tieto Czech s.r.o., s čímž jsem byl velice spokojen.

Mým úkolem byla implementace Unit Testů na jeden z existujících projektů. Dalším úkolem bylo generování dokumentace ze zdrojových kódů stejného projektu. Následně jsem se opět vrátil k původnímu úkolu unit testů, kde se mezitím vynořily skryté problémy a další možnosti pro vylepšení.

2 O firmě a mém zařazení

2.1 Tieto

Tieto je původem finská firma zaměřující se na komplexní služby v oblasti IT. Orientuje se především na severní Evropu, ale i na další trhy dohromady dvaceti zemí světa.

V české republice působí od roku 2001 a v roce 2012 má již téměř 2000 zaměstnanců. V našem kraji je tak největším zaměstnavatelem v oblasti IT.

2.2 Tieto v Ostravě

Během mé praxe se odehrála poměrně důležitá změna týkající se Ostravské pobočky. Doposud se firma rozkládala ve čtyřech různých budovách po celé Ostravě. V říjnu 2012 měla být pro Tieto dokončena stavba nové budovy v centru Ostravy a měl jsem nastoupit rovnou na novém pracovišti. Její dokončení se však o měsíc protáhlo a tak jsem zakusil staré i nové pracoviště.

Stěhování se týkalo téměř všech zaměstnanců s výjimkou některých týmů pracujících v technologickém parku Vysoké Školy Báňské. Přemístit téměř dva tisíce lidí není jednoduchý proces a trval zhruba do konce roku 2012. Nadále se ovšem dějí další přemístění v rámci budovy s cílem ideálního rozložení týmů.

2.3 Pracovní zařazení

Byl jsem zařazen k týmu u týmu NX na projektu Online. Jedná se o tzv. Nordic Product Portfolie, tedy produkty severských zemí, týkající se energetiky. NX Online je částí tohoto systému a stará se o webové prostředí pro koncové zákazníky a call centra energetických společností. Tým čítá aktuálně sedm členů a jednoho testera. Na některých úkolech jsem pracoval se svým spolužákem Dominikem Moravcem, který zde také absolvoval praxi.

3 Zadané úkoly

3.1 Unit testy

Úkolem bylo na stávající webový projekt naimplementovat a vytvořit sadu unit testů.

3.1.1 Co jsou to unit testy

Unit testy, to je metodika testování. Spočívá v rozdělení aplikace (systému) na jednotlivé částí, unity, a v jejich testování pokud možno nezávisle na ostatních částech aplikace. Obvyklé se testy dělí na třídy a testují se její jednotlivé metody. Jejich funkčnost se obvykle ověřuje na základě předpokládaného výsledku.

3.1.2 UT scénáře

Pro použití UT potřebujeme mít sadu vstupních a výstupních dat, u kterých víme, že pro testovanou metodu fungují správně. Čím větší množství dat máme, tím bude test účinnější. Dalším úkolem tedy bylo přepracovat testovací stránku aplikace tak, aby skrze ni bylo možné tato data, neboli scénáře, nahrávat.

3.2 Fakebackendsystem - FBS

3.2.1 Proč potřebujeme FBS

Protože se unit testy psaly nad nejvyšší fasádní vrstvou a webová aplikace se přistupuje k databázi, je nutné zajistit pro unit testy konzistentní data. Pro tyto účely používáme jejich nahrání do falešné databáze, která se zaměňuje se skutečnou.

FBS má i další využití, kromě UT:

- Pro opakované procházení scénářů při vývoji a testování. Například při mazání zákazníka by se musel pokaždé znovu vytvořit. Pokud ovšem při prvním smazání proces nahrajeme do FBS, můžeme jej opakovat znova a znova.
- Pro nahrání dat pro následnou analýzu při nalezení chyby u zákazníka.
- Pro urychlení načítání dat z databáze.

3.2.2 Jak funguje FBS

Aplikace se v provozu připojuje na vzdálené databáze. Tato volání se provádějí pomocí XML a stejnou formou se výsledky vracejí. Každý takovýto dotaz lze uložit do naší falešné databáze a při příštím dotazu se z naší databáze vybere stejná odpověď jako při jejím nahrávání.

3.3 Generování dokumentace ze zdrojového kódu

3.3.1 Analýza

Mým úkolem bylo udělat analýzu možných dokumentačních nástrojů a společně s kolegy vybrat ten nejvhodnější. Základní kritérium bylo pravidelné automatické generování webové stránky s dokumentací C# kódu více projektů zároveň.

3.3.2 Implementace na TFS

Protože jsou zdrojové kódy uloženy na společném serveru, kde se synchronizují mezi programátory, bylo nutné na tomto serveru vytvořit automatické sestavení dokumentace v nočních hodinách, kdy se neprováděly změny v kódu.

3.3.3 Dokumentace

Neboť v týmu není nikdo, kdo by se problematice věnoval, jsem jediný, kdo vytvořené práci rozumí a dokáže ji spravovat. Je tak velice důležité zdokumentovat mou práci a sepsat dokumentaci, jak jsem postupoval a jak vše funguje.

3.3.4 Ukázkové komentáře

Další část úkolu zahrnovala ukázkové okomentování kódu, který byl napsán v rámci UT a FBS. Toto mělo sloužit jako ukázka možností a správnosti komentování. Dále jsem vytvořil podrobný návod, jak tyto komentáře psát, aby byly efektivní, a jak je využívat. Tyto znalosti jsem také předal svým kolegům v prezentaci.

3.4 MVC Energy Reporting

Poté, co jsme dokončili naše úkoly, zbýval ještě necelý měsíc do konce našeho působení. Proto bylo mně a mému spolužákovi zadáno přepsání jednoho z modulu aplikace z ASP.NET WinForms do MVC.

4 Popis produktu NX Online

4.1 Rozvrstvení aplikace

Ke snadnému a efektivnímu napsání UT velice pomohla architektura, kterou byla aplikace napsaná. Ta byla rozdělena na více vrstev. Základem jsou klasické tři vrstvy: Business layer, Data layer a View layer. Takto se dělí do tří projektů, kde každý z nich zastupuje jednu vrstvu. Ovšem i tyto projekty jsou často obaleny další vrstvou pro snadnější záměnu těchto modulů a snadnější a přehlednější volání jejich metod.

4.1.1 Viewlayer

V našem případě se jedná o webovou aplikaci. Některé její části jsou již napsány v MVC, jiné zůstávají ve formě webové formuláře s hojně využitým JavaScriptem.

V této části aplikace jsem se moc nepohyboval. Nachází se zde ale také testovací webový formulář, který je využíván k nahrávání UT a FBS scénářů, respektive k zapnutí tohoto nahrávání v UI.

4.1.2 Business layer

Jejím úkolem je zprostředkování a zpracování dat mezi datovou a pohledovou vrstvou a obsahuje hlavní logiku aplikace.

Pro přístup k business vrstvě je zde použit návrhový vzor Facade, tedy fasáda. Ta slouží jako interface pro každou komponentu business logiky a zjednodušuje její volání. Je to vlastně jediná třída business logiky, která je viditelná zvenčí.

Právě zde, na fasádě se odehrávají unit testy, protože je to nejvyšší vrstva před uživatelským rozhraním a testuje se tak funkčnost celé datové části systému.

4.1.3 Data layer

Datová vrstvá se nestará přímo o uchování dat. Je to jen další vrstva mezi databázovými systémy, které běží odděleně a business logikou pro snadnější přístup k datům.

Obsahuje jednotlivé kontroléry (Controls), které umí pracovat s potřebnými databázemi. Tyto databázové servery komunikují pomocí XML, najdeme zde tedy další komponenty pro práci s XML, ale také Transfer Objects, které reprezentují třídy objektů posílaných na databáze a zpět.

Na této vrstvě pracuje FBS, protože je to poslední místo, kde se pracuje s daty, než se posílají a přijímají z databáze. Proto je toto ideální místo k zachycení dat do naší falešné databáze. Díky komunikaci v XML se toto zachytávání velice zjednodušuje.

4.1.4 Common

V Common se nacházejí třídy, které jsou pro celý projekt společné. Patří zde například Session a Cache manažery, Log manažer, manažer konfigurací a nebo výjimky.

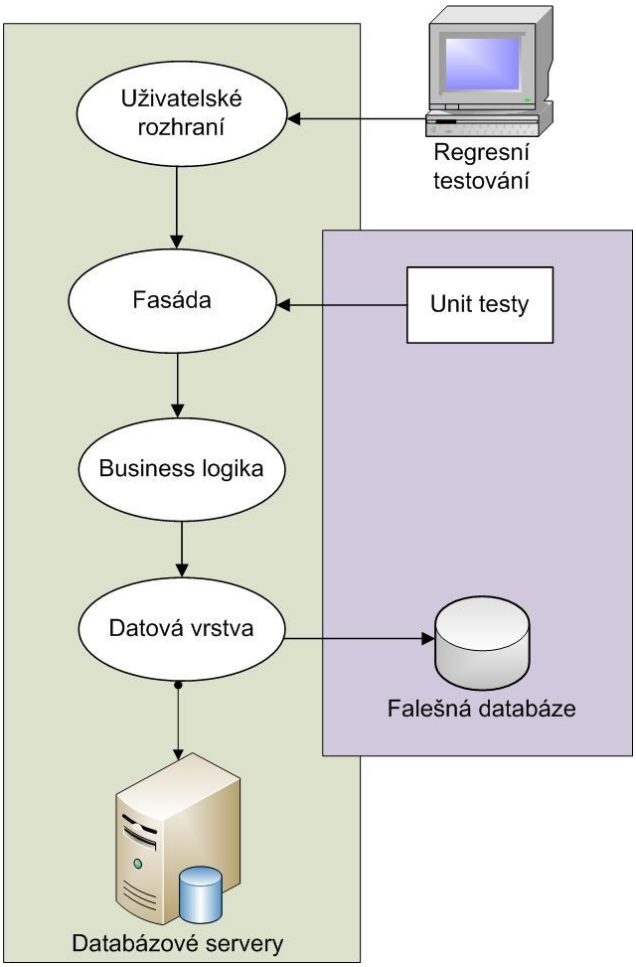
4.2 Návrhový vzor Továrna

Velice užitečný byl tento návrhový vzor, který byl implementován u většiny tříd. Hlavně díky němu jsme nemuseli aplikace celou přepisovat a neměnili jsme nijak její architekturu. Její architekti mysleli na budoucí úpravy a vytvořili prostředí tak, aby se jednotlivé části programu daly nahrazovat nejen na úrovni vrstev, ale až na úrovni tříd.

Princip je následující: třída, která obsahuje nějaké metody, není volána přímo. Místo toho je napsán interface, který tyto metody obsahuje, a také je vytvořen manažer, který je povětšinou statický a nabízí stejnou sadu metod. Máme tedy interface, statickou třídu a původní třídu, pokaždé se stejnou sadou metod.

Manažer se nyní stává branou k naší třídě. Po zavolání metody je zvolena požadovaná třída, která musí implementovat náš interface. Po správném výběru se na tuto třído volá stejná metoda, jaká byla volána na manažera.

Ve výsledku tak můžeme mít tři třídy, kdy každá bude mít stejný interface, ale každá dělá trochu něco jiného. Například jedna ukládá data do souboru, druhá do databáze a třetí na internet. A teprve za chodu programu se zjistí dostupnost databáze a internetu a zvolí se správná třída. Manažer se ve většině případů rozhoduje na základě konfigurací, nebo nastavení session.



Obrázek 4 .1: Obrázek 1

5 Implementace UT

5.1 Integrace

Právě díky zmíněnému vzoru továrny byla otázka integrace předem vyřešená. Pro svou potřebu jsme vytvořili kopie všech fasádních tříd a zaměnili je s těmi původními. Rozhodovací logika v manažerovi je nastavena na session, která se nastavuje v testovacím UI pomocí check boxu. Stejným způsobem v UI se nastavují také detaily nahrávání, tedy zda se mají přepisovat existující data a volitelný komentář. Velkou výhodou je i to, že pokud nebudou změněny žádné konfigurace, neprojeví se žádná ze změn v kódu, který je určen zákazníkovi.

5.2 Zachytávání dat

Přesto, že cílem bylo nahrávat konkrétní sadu metod, byl vytvořen univerzální Framework, který lze použít na libovolnou metodu, nejen fasádní. Výjimkou jsou konstruktory a metody bez návratové hodnoty. Byl kladen důraz na jednoduchost použití a výsledkem je jediná statická metoda, kterou je nutné volat v nahrávané metodě těsně před navrácením návratové hodnoty.

Pokud by původní metoda vypadala takto:

```
public CustomerTO GetCustomer(string code)
{
      return CustomerBE.GetCustomer(code);
}
```

Upravená verze vypadá takto:

```
public CustomerTO GetCustomer(string code)
{
var result = CustomerBE.GetCustomer(code);
UnitTestFramework.SaveData(code, result);
return result;
}
```

5.3 UnitTestFramework.SaveToTable(object[])

SaveToTable je jediná metoda celé třídy, která je vidět zvenčí. Její úkol je zapsat všechny vstupní parametry metody a výsledek, který vrací, a připojit použitou konfiguraci a také případný komentář.

Při použití metody je velice důležité, aby její parametry byly definovány všechny a ve správném pořadí. Je nutné předat ji postupně všechny vstupní parametry ve stejném pořadí, v jakém byly předány volající metodě a jako poslední zadat proměnnou, která bude vrácena.

Důvodem je způsob, jakým se vytvářejí databázové tabulky. K tomuto se používá reflexe, tedy metoda, která umožňuje zkoumat a manipulovat s již běžícím procesem a jeho typy. Takto získám nejprve objekt reprezentující metodu, která je ukládána, stejně jako její název a názvy všech jejích parametrů. Z tohoto se dále sestavuje SQL dotazy pro vytváření a plnění tabulek. Název tabulky se

shoduje s názvem metody a je dále doplněn o číslo, které určuje počet parametrů. To je zde kvůli přetíženým metodám, které mohou mít stejný název, ale různý počet parametrů. Každý parametr má v tabulce svůj sloupec se svým názvem, přidán je také sloupec „expectedResult“, do kterého se ukládá výsledek metody, a nakonec dva sloupce pro zmíněný komentář a pro název konfiguračního souboru.

Pro uložení dat do tabulky používáme serializaci. Použili jsme knihovnu .NET Frameworku, pouze zabalenou do třídy, která zjednodušuje její použití. Výsledek serializace je ve formátu XML, tedy prostý text a pro uložení do databáze je proto zvolen datový typ nvarchar(MAX).

5.4 Jak se píší Unit testy

5.4.1 Výběr technologie

Pro tento úkol byl zvolen Framework Microsoftu. Jako vývojové prostředí se v Tietu používá Visual Studio a jako takové již tento Framework obsahuje včetně pokročilé integrace a debugeru. Nabízí také předgenerování testovací třídy a psaní testů se takto velice usnadní. Ve vyšší verzi Visual Studia navíc poskytuje funkci Code-coverage, tedy v překladu pokrytí kódu. Tato funkce na konci testu dokáže barevně vyznačit části kódu, který byl, nebo nebyl otestován.

5.4.2 Generování UT a základní testování

Visual Studio nabízí velice jednoduchý způsob jak vygenerovat testovací třídu. Stačí pouze kliknout na testovanou třídu pravým tlačítkem a z kontextové nabídky zvolit „generate Unit Test class“. V dialogovém okně si vybereme ze seznamu metody, které budeme testovat, a po potvrzení se vytvoří projekt obsahující kostru naší testovací třídy.

```
[TestClass]
public classCalculatorTests
{
  [TestMethod]
  public voidPlusTest()
  {
    // Arrange: Createan instance to test:
    var calculator = newCalc();

    // Act: Run themethodunder test:
    double result = calculator.Plus(0,0);

    // Assert: Verifytheresult:
    Assert.AreEqual(0+0, result);
  }
}
```

Takto by vypadal jednoduchý unit test pro kalkulačku. Kalkulačka by měla svou testovací třídu a ta by obsahovala testovací metody pro jednotlivé funkce jako sčítání, nebo odčítání. Čísla se mohou například generovat náhodně, nebo načítat z pole. Provede se operace, kterou testujeme, tedy sčítání, a porovná se, zda je výsledek takový, jaký se očekává. Pokud ano, test se označí jako úspěšný, v opačném případě selže a my si jej můžeme odkrokovat, abychom zjistili, kde je problém.

5.4.3 Načítání dat z databáze

Testovat metodu tímto způsobem by bylo velice pracné. Naše metody mají v parametrech složité objekty, které by se musely vytvořit a naplnit. Naštěstí můžeme použít naši databázi a použít skutečná data, která jsme si nahráli ze skutečného systému. Navíc přesně víme, jaký má být výsledek operace.

Pro načítání dat z databáze nemusíme psát vlastní kód, ale můžeme využít funkce Frameworku. Testovací metody podporují atributy pro získání dat z tabulky a její obsah nám servíruje po řádku v proměnné TestContext. V atributu metody tak nastavíme connectionstring, tedy řetězec určující umístění databáze, způsob přihlášení a providera a nakonec název tabulky.

```
[TestMethod()]
[DataSource("System.Data.SqlClient",
     "Data Source=.\\SQLEXPRESS;AttachDbFilename=UnitTestsDb.mdf;,
     "Calc_Plus_2",
DataAccessMethod.Sequential)]
public voidPlusTest()
{
 var calculator = newCalc();

 doublea = Convert.ToDouble(testContextInstance.DataRow["a"]);
 doubleb = Convert.ToDouble(testContextInstance.DataRow["b"]);

 double x = Convert.ToDouble(testContextInstance.DataRow["x"]);

 double result = calculator.Plus(a,b);

 Assert.AreEqual(x, result);
}
```

Protože naše data nejsou uložena přímo, musíme je ještě před použitím deserializovat do objektu. Pro tento účel jsem napsal generickou metodu, která na základě názvu proměnné a datovém typu vrací požadovaný objekt načtený z databáze. Pro zjednodušení nejsou na konci porovnávány dva objekty – tedy výsledek testu a deserializovaný výsledek z databáze – ale raději serializuji výsledek a porovnám jej na úrovni řetězce s řetězcem z databáze.

6 Implementace FBS

6.1 Integrace

I zde nám pomohl zmíněný návrhový vzor Továrna. Na nejnižší vrstvě aplikace, tedy datové, se nacházejí adaptéry pro jednotlivé databáze, se kterými aplikace komunikuje během svého chodu. Všechny tyto adaptéry se používají pomocí manažerů. V tomto místě se proto změnila logika výběru modulu. I v tomto případě se využilo testovacího uživatelského rozhraní a checkboxů, kterými se dá zapnout a vypnout nahrávání. Z toho vyplývá, že kromě originálního modulu byly napsány další dva. Pro nahrávání a načítání dat. Další možnost rozhodování je na základě konfiguračních souborů.

FBS v testovacím UI nabízí další volby. Je možné vybrat si databázi, přepisování existujících dat a nakonec přibyla také možnost vytvářet a spravovat skupiny. Ty jsou užity pro lepší organizaci nahraných dat. Například která jsou určená pro unit testy a která pro ruční testování.

6.2 Ukládání dat

Protože naprostá většina této komunikace probíhá pomocí XML, nebylo zde nutné psát žádné speciální funkce. Vytvořil jsem proto klasického datového providera, který poskytuje ukládání a načítání z databáze a umožňuje vytvoření tabulky. Společně s daty, tedy vstupní a výstupní XML, se do databáze ukládá také HASH verze vstupního XML pro urychlení dotazů, název konfigurace, modul, kterým byly data pořízena, a organizační skupina. Ukládající modul je s originálním prakticky identický, pouze ve správný okamžik uloží data do databáze.

6.3 Načítání dat

Načítání je vyřešeno stejně jako ukládání, ale těla některých metod jsou nahrazena voláním falešného providera. Ten místo skutečné databáze posílá dotazy na falešnou databázi a vrací dříve uložené záznamy.

7 Generování dokumentace

7.1 Výběr produktu

Mým prvním úkolem bylo udělat průzkum možností. Zjistit jaké nástroje existují, jaké mají možnosti a co dokáží. Měl jsem sestavit přehled jejich funkcí a vlastností a samozřejmě také jejich cen. Výsledek tohoto průzkumu jsem prezentoval celému týmu a po společné diskuzi a mém doporučení jsme vybrali open-source projekt SHFB.

Dalším krokem po výběru byl průzkum. Podrobněji jsem zjišťoval, co vše SHFB dokáže, jaké přináší možnosti, vyzkoušet si je a vytvořit ukázky generované dokumentace včetně pokročilejších funkcí. Na základě těchto zjištění jsme si ujasnili, jaký přesně požadujeme výsledek.

7.2 SHFB a Sandcastle

SHFB je zkratkou Sandcastle Help File Builder a jedná se o nástroj, který pomáhá při tvorbě dokumentace ze zdrojového kódu pomocí nástroje Sandcastle.

Sandcastle je původní nástroj Microsoftu, který je používán ke generování dokumentace a je pomocí něj vytvářena například nápověda na webu MSDN. Ovládá se ovšem výhradně v příkazové řádce a postrádá pokročilé funkce.

Proto vznikl open source projekt s názvem SHFB, který výrazně rozšířil možnosti Sandcastle a nabídl uživateli grafické uživatelské rozhraní. Novější verze již vytvářejí projekty kompatibilní s projekty Visual Studia a může díky tomu být součástí solution a být sdílen pomocí TFS. Umožňuje také různé styly výstupu, jako je soubor standartních nápověd ve Windows, nebo webová stránka v ASP.NET. Další výhodou novější verze je integrace v prostředí Visual Studia pomocí pluginu.

Projekt byl po čase přesunut pod křídla Microsoftu, ale i přesto se netěší příliš časté aktualizaci.

7.3 Komentování zdrojového kódu

Součástí mé prezentace bylo zaškolit své kolegy do možností, které nabízí komentovaný kód. Nejprve jsem tedy ukázal, co vše tím získají – kromě výsledné webové nápovědy také integraci nápovědy do IntelliSense, včetně popisu parametrů, a možnost strukturovaně procházet objekty s popisem v ObjectView okně ve vývojovém prostředí.

Dále jsem předvedl, jak se metody a třídy komentují a jaké tagy Sandcastle, respektive SHFB, nabízí. Doporučil jsem používat především základní tagy, které VS generuje na základě definice funkce. Použití je velice jednoduché, protože stačí na řádek před danou metodou nebo třídou napsat tři zpětná lomítka. VS vygeneruje kostru komentáře ve formátu XML, který je na začátku každého řádku vždy doplněn o tři zpětná lomítka aby byl zřetelně rozpoznatelný pro nástroje, které s ním pracují.

Součástí kostry je vždy sekce Summary, kde má autor metody (třídy) ve zkratce shrnout její funkci a princip. Tento text se zobrazuje v IntelliSense. Doplňující informace se vkládají do pole Remarks. To se již v IntelliSense neobjevuje, patří zde proto detailní informace jako jsou například příklady použití, nebo doplňující teorie. Oba tyto prvky nabízejí navíc paragrafy pomocí tagů Param.

Velice užitečné je komentování parametrů. Pokud metoda nějaké má, automaticky se generují také v komentáři. Během použití metody se tento popis zobrazuje u jednotlivých parametrů a programátor nemusí přemýšlet, co má proměnná představovat a jaký má účel.

Nakonec sem patří návratová hodnota, která zahrnuje i shrnutí toho, co nám funkce vrací. Taktéž se zobrazuje v IntelliSense a často se význam této věty podobá poli Summary.

Na výběr pak máme ještě poměrně velkou škálu tagů, ovšem jejich použití již není natolik efektivní a usoudil jsem, že je vhodnější programátory nezatěžovat přílišnou složitostí a neodrazovat je tak. Mezi tyto tagy patří například See a SeeAlso, které vygenerují odkaz na jinou část dokumentace, nebo C, respektive Code, který slouží k zobrazení zdrojových kódů včetně barevného zvýraznění. Protože náleží mezi automaticky generované, často se vyskytne také Exception, který popisuje možné výjimky.

7.4 Implementace

Běžně stačí přidat projekt dokumentace do solution aplikace, kterou má dokumentovat, a přidělit ji referenci na solution soubor. V našem případě bylo ale nutné zvolit jinou cestu. Dokumentace se má generovat pravidelně, a to v nočních hodinách z aktuální vývojové verze aplikace, která je umístěna na TFS. Protože slouží také jako buildovací server, umístil jsem celý dokumentační projekt rovněž na TFS. Pravidelné sestavování je pouze položka v nastavení. Bylo ale nutné napsat sestavovací skript, který vykoná všechny kroky. Tento skript je opět tvořen XML tagy a určuje scénář sestavení, tj. určuje, jak se budou jednotlivé operace vykonávat.

Postup sestavení tedy začíná sestavením dokumentované aplikace. Tím získáme soubory potřebné k sestavení dokumentace, tedy XML s komentáři a dll knihovny. Tyto soubory jsem načetl do pole a později předal jako parametr sestavení dokumentace. Po jejím sestavení se výstupní soubory zabalí a jsou poslány a následně rozbaleny na jiném serveru, kde bude webová stránka uložena. Zde je již nakonfigurován ISS pro správnou reprezentaci webu pomocí webové adresy.

8 Energy Reporting MVC

Na tomto úkolu pracujeme v okamžiku odezvdání této práce a nevím, jak velkou část stihneme dokončit. Proto u tohoto konkrétního úkolu popíši jen zadání a popis, nikoli celé jeho řešení.

8.1.1 Návrh Model – View – Controll

Původní modul byl napsán pomoci ASP.NET WebForms a bylo zde využito velké množství JavaScriptu, ve kterém se dalo těžko vyznat. Stejně jako jiné moduly, byl i tento určen k přepsání do modernější architektury MVC. Ten rozděluje aplikaci na tři vrstvy, kdy View je webová stránka ASP.NET s použitím Razor syntaxe, která obohacuje HTML syntaxi o C# kód. Díky tomu se může pohled měnit v závislosti na modelu. Ten reprezentuje data, která mají být zobrazena uživateli. A controller se stará o to, aby tato data získal. Controller funguje jako zprostředkovatel mezi webovým prohlížečem a serverem, který poskytuje obsah.

8.1.2 Graf jQPlot

Modul, který jsme přepisovali, se nazývá Energy Reporting a jeho úkolem je zobrazovat statistiky naměřených dat o spotřebě energie, cenách a podobně. Dominantním prvkem na stránce je tedy samotný graf a dále ovládací prvky pod ním. Těmi uživatel rozhoduje, co se na grafu zobrazí.

Tyto ovládací prvky jsou součástí několika formulářů. Pokud je nutné překreslit graf, vyšle se společně s informací o zákazníkovi také aktuální nastavení formuláře na server, tedy controlleru. Controller z tohoto formuláře vytáhne všechny podstatné informace a na jejich základě získá z databáze data pro vykreslení grafu.

Následně je nutné tato data zanalyzovat a vhodně nakonfigurovat graf. Tedy nakonfigurovat vhodné rozlišení os, legendu a podobně. Pro tento účel jsme vytvořili strukturu tříd odpovídající třídám konfiguračního souboru a vytvořili k němu Serializer, který tento objekt převede do takové textové podoby, které rozumí plugin použitý pro graf.

9 Získané znalosti a zkušenosti

9.1 UT a FBS

Celé mé působení ve firmě pro mne bylo velice přínosné. Od počátku pro mne bylo zajímavé zapojit se do koloběhu velké firmy a sledovat firemní procesy. Na začátku bylo mým úkolem nejprve zorientovat se ve firmě, firemním intranetu, a následně si nastudovat látku týkající se našeho úkolu. K danému tématu jsem měl k dispozici knihu, ze které jsem mohl čerpat.

Asi nejtěžší bylo naučit se a pochopit, jak aplikace, na které jsme pracovali, funguje. Přesto, že její architektura není tak složitá, nebylo jednoduché udělat si pořádek v obrovském množství tříd a zároveň si dávat pozor, abych nepoškodil nějakou její funkční část.

Protože s programováním v C# mám zkušenosti ze své domácí činnosti i ze školy, neměl jsem s ním problém. Méně znalý jsem byl pouze architektuře WebForm a musel jsem si převyknout na trochu odlišný styl oproti desktopovým aplikacím ve WPF.

Jsem také velice rád, že jsem se seznámil a osvojil si mnoho dobrých zvyků a praktik od zkušenějších programátorů.

9.2 Dokumentace

Během práce na UT a FBS jsem se věnoval dokumentačním nástrojům. V této oblasti byly mé znalosti chabé a má práce byla zaměřena spíše na výzkum, testování a objevování nového. Přesto že pro mne nebyla tato část tolik zábavná, byla velice přínosná.

Díky práci s komentáři a jejich prezentování psanou i písemnou formou ostatním jsem si sám osvojil dobrý zvyk komentovat metody a třídy. Během této praxe jsem sám na sobě zpozoroval, že jsem začal psát komentáře dokonce i na svých soukromých projektech.

Dále jsem se naučil kompletně ovládat dokumentační nástroj SHFB.

Velice rád jsem se naučil principy práce s TFS, pochopil jsem, jak tento systém funguje a vyzkoušel si napsat vlastní sestavovací skript. Tato činnost by mi mohla být v budoucnu velkým přínosem.

Protože je firma Tieto mezinárodní korporací, mnohá komunikace probíhá v angličtině. Krom běžného používání jsem si mohl vyzkoušet v anglickém jazyce i prezentace. Takovou příležitost jsem měl poprvé mimo půdu školy.

10 Závěr

S celkovým průběhem této praxe jsem velice spokojen a jsem rád, že jsem se pro ni rozhodl. Během mého působení jsme vytvořili systém, který byl předmětem naší praxe, a navíc jsem v mezičase pracoval na dokumentaci. Ve chvíli, kdy dopisuji tento dokument, stále pracuji na doladění drobných detailů, které se projevují až během provozu. Získal jsem mnoho zkušeností a znalostí a také jsem si ujasnil své další životní a kariérní cíle.

Použitá literatura

[1] Larry Brader, Howie Hilliker a Alan Wills. MICROSOFT. *Testingfor Continuous Delivery with Visual Studio 2012*. ISBN 978-1-62114-019-1.

[2] Dokumentace projektu Sandcastle Help File Builder. *shfb.codeplex.com* [online]. Dostupné z: http://shfb.codeplex.com/documentation/

[3] MICROSOFT. MSDN: *Microsoft Developer Network* [online]. Dostupné z: http://msdn.microsoft.com/